# Utilisation de python pour le calcul numérique

**Résumé**

L'objectif de ce TP est de découvrir quelques possibilités de python pour le calcul numérique. Il pourra également vous servir de référence si vous avez besoin de faire du calcul numérique en TIPE.

Il existe trois librairies en python permettant de résoudre la plupart des problèmes numériques auxquels on peut être confronté :

— La librairie NumPy implémente des tableaux de données.

— La librairie SciPy, qui s'appuie sur NumPy implémente de nombreuses fonctions de calcul numérique (résolution d'équations, calcul d'intégrales, intégration d'équation différentielles, régression linéaire, etc.)

— La librairie matplotlib permet de tracer des graphes.

## 1 Rapide introduction aux tableaux

■ Recopier et exécuter les lignes de code ci-dessous.

```
1 import numpy as np
2 tableau = np.array([[0,1,2],
3                     [1,2,3],
4                     [4,6,12],
5                     [44,55,56]])
6 n = tableau[3,0]
7 c = tableau[:,0]
8 print(n)
9 print(c)
```

La ligne 1 permet d'importer la librairie NumPy qui contient de nombreuses fonctions permettant de travailler avec des tableaux. À la ligne 2, on appelle ainsi la fonction `array` de NumPy sous la forme `np.array`.

⚠ `from numpy import *` importe toutes les fonctions de la libraire, ce qui peut ralentir le programme. De plus, si une fonction `array` avait déjà été définie, elle est écrasée.

Il existe deux autres manières d'importer une librairie :

— `import numpy`, on appelle alors la fonction `array` avec : `numpy.array`,

— `from numpy import *`, on appelle alors directement la fonction `array` avec : `array`.

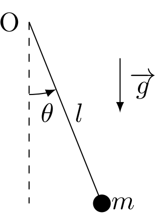
■ Déterminer la syntaxe permettant d'afficher le nombre en quatrième ligne et deuxième colonne du tableau (55) et celle permettant d'afficher la troisième ligne du tableau (qui contient 4, 6, 12).

Deux fonctions de NumPy sont utiles pour la suite : `arange` et `linspace`.

■ Que fait le code ci-dessous ?

```
tableau_1 = np.arange(1,10,.3)
print(tableau_1)
tableau_2 = np.linspace(1,10,15)
print(tableau_2)
```

Nous allons nous intéresser dans la suite au mouvement d'un pendule simple : une bille M de masse $m$ est suspendue à une tige rigide OM de longueur $l$ et de masse négligeable. Le pendule est libre d'osciller dans le champ de pesanteur terrestre $\overrightarrow{g}$ ($g = 9,81\ \mathrm{m \cdot s^{-2}}$).




## 2 Résolution de systèmes d'équations linéaires avec NumPy

Le code ci-dessous résout le système d'équations :

$$\begin{cases} x + 2y = 3 \\ 3x - 4y = 1 \end{cases}$$

```python
systeme = np.array([[1,2],
                    [3,-4]])
valeur = np.array([[3],
                   [1]])
sol = np.linalg.solve(systeme,valeur)
print("x=",sol[0,0]," et y=",sol[1,0])
```

En présence de frottements importants, on peut montrer que le mouvement du pendule est donné par

$$\theta(t) = Ae^{r_1 t} + Be^{r_2 t}$$

Si l'on lâche le pendule d'un angle $\theta_0$ sans vitesse initiale, A et B sont alors solutions du système :

$$\begin{cases} A + B = \theta_0 \\ r_1 A + r_2 B = 0 \end{cases}$$

■ On donne $r_1 = -0,316$ s$^{-1}$ et $r_2 = -31,0$ s$^{-1}$. Déterminer les valeurs numériques de A et B pour $\theta_0 = 0,2$ rad.

# 3 Résolution d'équations avec SciPy et matplotlib

## 3.1 Cas d'une solution unique

La fonction `brentq` contenue dans la librairie SciPy permet de déterminer numériquement les racines d'une équation quelconque, on peut s'en servir pour résoudre une équation de la forme $f(x) = 0$. Le code ci-dessous résout l'équation $(\ln x)^x = 3$, soit $(\ln x)^x - 3 = 0$.

⚠ Il est important lorsqu'on utilise SciPy de travailler avec les fonctions usuelles (sin, ln, etc.) de NumPy ou de SciPy, conçues pour travailler avec des tableaux. **Il ne faut surtout pas utiliser celles de la librairie `math`** vue dans un TP précédent qui ne sont pas conçues pour travailler avec des tableaux et peuvent mener à des résultats faux. C'est pour cela qu'on utilise `from scipy import *`.

```python
from scipy import * #Pour utiliser les fonctions mathématiques de scipy
from scipy.optimize import brentq #Pour utiliser directement brentq
def f(x):
    return log(x)**x-3
sol = brentq(f,1,100)
print("racine pour x=",sol)
```

L'instruction `from scipy import *` permet d'importer toutes les fonctions de SciPy, entre autres les fonctions usuelles comme `log`, qui correspond au logarithme népérien (`log10` correspond au logarithme en base 10), ainsi que toutes les fonctions de la librairie NumPy qui est incluse dans SciPy.

La documentation de la fonction brentq sur le site internet de scipy précise son mode d'emploi.

■ D'après la documentation, quelle est la signification des trois arguments de la fonction `brentq` ?

Revenons au pendule, on veut déterminer l'instant $t$ pour lequel $\theta(t)$ vaut $\theta_0 = 0,1$ *rad*. Dans le cas général, l'équation $\theta_0 = Ae^{r_1 t} + Be^{r_2 t}$ n'a pas de solution analytique : on doit la résoudre numériquement.

■ Adapter le code proposé pour déterminer l'instant pour lequel $\theta(t)$ vaut $\theta_0 = 0,1$ rad. (Utiliser les valeurs de A et B déterminées précédemment).

## 3.2 Cas de solutions multiples

Quand une équation a plusieurs solutions sur un domaine donné, `brentq` n'en retourne qu'une seule, qui dépend des paramètres de l'algorithme de résolution. Pour déterminer les différentes solutions d'une équation, on peut s'appuyer sur une étude graphique de la fonction associée.

Le code ci-dessous trace l'allure de $\sin x + x^2$ pour $x \in [0, 2]$ :

```python
from scipy import *
import matplotlib.pyplot as plt #Permet de tracer des graphes
x = linspace(0,2,50)
y = sin(x) + x**2
plt.plot(x,y)
plt.show()
```

Revenons au pendule, si l'amortissement n'est pas trop important, il oscille et son mouvement est de la forme $\theta(t) = \theta_m e^{-t/\tau} \cos(\omega t + \varphi)$. On s'intéresse à un cas particulier où $\theta_m = 0,2$ rad, $\tau = 3$ s, $\omega = 3$ rad · s$^{-1}$ et $\varphi = 0$.

■ Tracer l'allure de $\theta(t)$ pour les premières oscillations.

On veut déterminer les trois plus petits instants positifs $t_1$, $t_2$ et $t_3$ pour lesquels $\theta$ vaut 0,05 rad.

■ Déterminer grossièrement à l'aide du graphe les intervalles dans lesquels sont contenus $t_1$, $t_2$ et $t_3$ et déterminer précisément leurs valeurs à l'aide de la fonction `brentq`.

# 4 Résolution d'équations différentielles avec SciPy

## 4.1 Résolution d'équations différentielles du premier ordre

SciPy permet d'intégrer (de résoudre) numériquement des équations différentielles. Le code ci-dessous permet de résoudre l'équation différentielle du premier ordre suivante : $y' = y \cdot sin(y) + \sin(x)$ avec $y(x = 0) = 2$.

```python
from scipy import *
from scipy.integrate import odeint #Pour résoudre l'équation différentielle
import matplotlib.pyplot as plt #Permet de tracer des graphes

def deriv(y,x):
    return y*sin(y) + sin(x)

#Choix de l'intervalle d'intégration
x0 = 0
xmax = 20
npoints = 1000
x = linspace(x0,xmax,npoints)

#Condition initiale
y0 = 2 # y(x0) = y0

solution = odeint(deriv,y0,x)

y = solution[:,0]

plt.plot(x,y)
plt.show()
```

On coupe la tige du pendule lorsqu'il est au repos. La bille de masse $m = 2,0 \times 10^{-2}$ kg, en polystyrène, subit alors une chute verticale sans vitesse initiale. En plus de son poids, elle subit un force de frottements de la forme $\overrightarrow{f} = -\alpha v \overrightarrow{v}$, avec $\alpha = 3 \times 10^{-2}$ $\mathrm{kg \cdot m^{-1}}$. En utilisant les lois de la mécanique, on montre que la vitesse vérifie l'équation différentielle :

$$\frac{\mathrm{d}v}{\mathrm{d}t} = g - \frac{\alpha}{m} v^2. \qquad (1)$$

■ Adapter le code précédent pour tracer l'évolution de la vitesse de la bille sur les 4 premières secondes de sa chute. Estimer le temps au bout duquel la bille atteint sa vitesse limite.

## 4.2 Résolution d'équations différentielles du deuxième ordre

On attache à nouveau la bille au pendule. En l'absence de frottements, les oscillations du pendule vérifient l'équation différentielle :

$$\ddot{\theta} + \omega_0^2 \sin\theta = 0 \qquad (2)$$

On prendra dans toute la suite $\omega_0 = 3$ $\mathrm{rad \cdot s^{-1}}$.

Dans le cas des petits angles $\theta \ll 1$, on a $\sin\theta \simeq \theta$, et l'on retrouve l'équation différentielle d'un oscillateur harmonique :

$$\ddot{\theta} + \omega_0^2 \theta = 0 \qquad (3)$$

L'équation 2 n'a pas de solutions analytiques, on se propose donc de l'intégrer avec SciPy pour comparer ses solutions à celles de l'équation 3. On prendra comme conditions initiales $\theta(t = 0) = 0,1$ rad et $\dot{\theta}(t = 0) = 0$.

Les algorithmes d'intégration numérique dont on dispose ne permettent d'intégrer que des équations différentielles du premier ordre.

Pour intégrer une équation différentielle du deuxième ordre, il faut la convertir en un système de deux équations différentielles du premier ordre. On pose $\theta_p = \dot{\theta}$, alors $\dot{\theta}_p = \ddot{\theta}$.

L'équation 3 est équivalente au système d'équations du premier ordre :

$$\left\{ \begin{array}{l} \dot{\theta} = \theta_p \\ \dot{\theta}_p = -\omega_0^2 \theta \end{array} \right. \qquad (4)$$

Ce système d'équations peut se résoudre avec SciPy à l'aide du code ci-dessous :

```
from scipy import *
from scipy.integrate import odeint

omega0 = 3#rad/s

def deriv(TH,t):
    # prend comme arguments :
    # TH : une liste contenant theta et thetap
    # t : un instant donné
    # retourne la liste : [thetap,-omega**2*theta]
    theta = TH[0]
    thetap  = TH[1]
    return [thetap,-omega0**2*theta]

#définition de l'intervalle de temps
t0   = 0
tmax = 15
npoints = 1000
t = linspace(t0,tmax,npoints)

#Définition des conditions initiales
theta0 = .1#rad
thetap0 = 0#rad/s
ci = [theta0,thetap0]

solution = odeint(deriv,ci,t)#la résolution proprement dite

theta = solution[:,0]
thetap = solution[:,1]
```

■ Recopier le code proposé et le compléter pour tracer l'allure de $\theta(t)$ pendant les 15 premières secondes des oscillations à l'aide de matplotlib.

■ Modifier le code pour calculer la solution de l'équation différentielle non linéaire (équation 2) avec les mêmes conditions initiales et pour tracer sur un même graphe les solutions des deux équations différentielles. Y a-t-il une différence importante entre les deux solutions ?

■ Choisir des conditions initiales correspondant à un lâcher avec un angle initial $\theta_0$ de 90°, puis avec un angle proche de 170°. Comparer les solutions de l'équation non-linéaire (équation 2) et de l'équation linéarisée (équation 3) dans les deux cas.

La librairie SciPy définit une variable `pi` qui contient la valeur de $\pi$.

■ Que se passe-t-il si l'on résout l'équation non-linéaire (équation 2) avec la condition initiale `theta0 = pi` ? Avec la condition initiale `theta0 = 3.14159` ? Comparer à la solution exacte de l'équation différentielle pour $\theta_0 = \pi$.

## 5 Calcul d'intégrales avec SciPy

Bien que l'équation différentielle 2 n'ait pas de solution analytique, il est possible de déterminer une expression théorique de la période $T$ du pendule. On peut montrer que :

$$T = \frac{2\sqrt{2}}{\omega_0} \int_{\theta=0}^{\theta_0} \frac{\mathrm{d}\,\theta}{\sqrt{\cos\theta - \cos\theta_0}} \qquad (5)$$

Cette intégrale n'a pas d'expression analytique simple. On se propose de la calculer numériquement avec python. On pourra prendre $\omega_0 = 3\ \mathrm{rad} \cdot \mathrm{s}^{-1}$.

L'exemple ci-dessous, tiré de la documentation de la fonction quad sur internet montre comment calculer l'intégrale : $\int_0^4 x^2 dx$.

```
from scipy.integrate import quad
def f(x):
    return x**2
integrale,erreur = quad(f,0,4)
print(integrale)
```

■ Adapter l'exemple proposé et calculer la période $T$ des oscillations pour $\theta_0 = 0,1$ rad, pour $\theta_0 = 3$ rad.

■ Comparer les résultats obtenus aux valeurs qu'on obtient pour le cas linéarisé (équation 3) : $T = \dfrac{2\pi}{\omega_0}$.

# 6 Fonctions spéciales

Au lieu de calculer numériquement l'intégrale, on aurait pu continuer un peu le calcul[1] et s'apercevoir qu'on peut aussi écrire :

$$T = \frac{4}{\omega_0} \int_{\phi=0}^{\pi/2} \frac{\mathrm{d}\,\phi}{\sqrt{1 - k^2 \sin^2 \phi}} = \frac{4}{\omega_0} K\left(k^2\right), \text{ avec } k = \sin\left(\frac{\theta_0}{2}\right) \tag{6}$$

où K est une fonction spéciale, appelée intégrale elliptique complète de première espèce (ou *complete elliptic integral of the first kind*).

L'avantage d'utiliser cette formule est que les fonctions spéciales comme celle-ci sont définies dans SciPy. L'implémentation de SciPy est généralement meilleure (plus rapide, plus précise, etc.) que celle qu'on pourrait proposer, on a tout intérêt à l'utiliser.

■ Utiliser la documentation de SciPy pour trouver la fonction correspondant à K et calculer la valeur de $T$ pour $\theta_0 = 0,1$ rad et $\theta_0 = 3$ rad. Comparer le résultat au calcul de l'intégrale.

# 7 Pour aller plus loin

■ Tracer la courbe donnant la période $T$ des oscillations du pendule simple en fonction de l'angle initial $\theta_0$.

Il existe une formule approchée donnant la période du pendule meilleure que la formule pour les petites oscillations ($T = \dfrac{2\pi}{\omega_0}$). Il s'agit de la formule de Borda :

$$T \simeq \frac{2\pi}{\omega_0}\left(1 + \frac{\theta_0^2}{16}\right). \tag{7}$$

■ Compléter le graphe précédent avec la période donnée par la formule de Borda.

■ Écrire une fonction `borda_juste(p)` qui retourne l'angle $\theta_{\max}$ tel que la formule de Borda donne le vrai résultat à moins de $p$ pourcent près pour $\theta < \theta_{\max}$.

1. On peut passer de la première intégrale à la deuxième en remarquant que $\cos\theta = 1 - 2\sin^2(\theta/2)$, puis en posant $k = \sin(\theta_0/2)$ et en introduisant $\phi$ tel que $k \sin\phi = \sin(\theta/2)$.